View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 2496 – 2499

Procedia Computer Science

2013 International Conference on Computational Science

Kernel performance improvement for the FEM-based fluid analysis code on the K computer

Kiyoshi Kumahata[a*], Shunsuke Inoue[a], Kazuo Minami[a]

[a]*Operations and Computer Technologies Div., RIKEN AICS, 7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan*

Abstract

The general purpose fluid simulation software FrontFlow/blue (FFB) is based on the finite element method (FEM). This software has two modes to handle pressure: one defines pressure on nodes and the other defines pressure on elements. For the element pressure mode, the gradient computation kernel is dominant to computation time. This kernel performs an operation in which a value is calculated using variables defined on an element and then stored in a node. Such an operation is frequently performed in FEM and the finite volume method. This kernel indirectly accesses memory via array values. Hence, improving memory/cache utilization is necessary to improve overall performance. This study estimates performance and tuning of FFB's gradient computation kernel on the K computer.

© 2013 The Authors. Published by Elsevier B.V. Open access under CC BY-NC-ND license.
Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science

Keywords: Fluid Analysis ; the K computer ; Finite Element Method ; Improving Performance

1. Introduction

RIKEN continues to work on improving the K computer's hardware and software. Software applications have been selected from a wide range of application areas. Two primary characteristics are considered when selecting appropriate software applications: parallelization techniques and single CPU performance. This study discusses improvements for one of the selected applications, the general purpose fluid simulation software, FrontFlow/blue (FFB). The rest of this paper is structured as follows. First, an overview of dominant computation kernel of FFB, and information on the K computer CPU are provided. Next, we discuss the ideal performance of dominant kernel of FFB on the K computer. Then, we present a method for improving performance and present measurement results.

* Corresponding author. Tel.: +81-78-940-5790; fax: +81-78-304-4955.
E-mail address: kuma@riken.jp.

1877-0509 © 2013 The Authors. Published by Elsevier B.V. Open access under CC BY-NC-ND license.
Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.427

2. FrontFlow/blue

FFB [1] is a general purpose computational fluid dynamics code for incompressible unsteady flow. It was implemented on the basis of the finite element method (FEM). Though FFB employs simple domain decomposition, massive parallelization can be easily achieved for large-scale problems. We have attempted single CPU performance tuning of the major computation kernel of FFB on the K computer. The most dominant kernel is the gradient computation for tetrahedral elements occupying approximately 30% of total computation time. This kernel operation is frequently occurred in the FEM and the finite volume method, and is important in engineering applications. Therefore, we conducted performance evaluations for the improvement of this kernel on the K computer.

3. Overview of the K computer CPU

The K computer, which was developed by RIKEN and Fujitsu, is a distributed memory supercomputer system with more than 80,000 compute nodes. It has 10 PFLOPS sustained performance for the LINPACK benchmark. The K computer is powered by a SPARC64 VIIIfx [3] CPU that has eight cores and a 6 Mbyte L2 cache that is shared by the cores. Its peak performance is 16 GFLOPS per cores. Each core has a 32 Kbyte L1D cache. Effective performance of the CPU is 123.6 GFLOPS (96.6% efficiency). Memory access performance has been measured at 46.6 Gbyte/s using the triad STREAM benchmark code [4].

4. Gradient computation kernel of FFB and its performance evaluation on the K computer

Fig.1 shows the kernel source code for a tetrahedron element. In this section, we estimate the ideal performance of the kernel on the K computer by the ratio of the required bytes of the kernel to the value of floating point operations [2], i.e., the B/F value. This estimation based on the amount of data transferred from memory having a dominant penalty as a required byte. By this way, the required bytes of the kernel is 4736 byte for 32 iterations of inner-most loop. And the number of floating operation is 768 for 32 iterations. Therefore, the B/F value can be determined as 4736/768 = 6.17. As mentioned in Section 3, the floating operation performance of the K computer CPU is 128 GFLOPS and the effective memory bandwidth is 46.6 Gbyte/s. This results in a B/F value of 46.6/128 = 0.36. Therefore, the ideal performance of the kernel on the K computer CPU is estimated at 5.83% (=0.36/6.17) of 128 GFLOPS. The actual measured performance ratio of the kernel was 1.6% and memory throughput is 10.29 Gbyte/s by test data it consists 820,000 elements. The theoretical value of L1D cache miss ratio is 3.125% on the K computer but in this result, the miss ratio was 21.3%. Therefore, ineffective cache access must be an obvious and significant bottleneck.

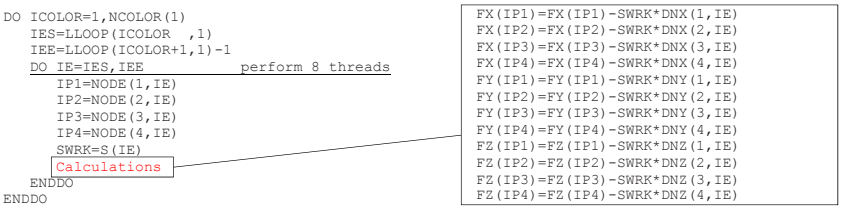


Fig.1. Source code

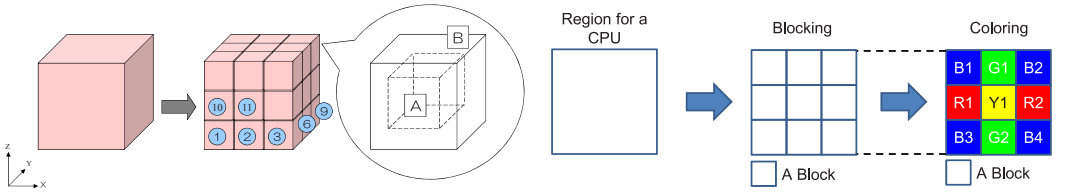
5. Tuning way and measurement result

A major cause of ineffective cache access is scattered memory access. Because array NODE values are non-sequential and have a wide range, memory access to arrays FX, FY, and FZ is scattered and few data is used on

the L1D cache line. Such array NODE value distribution results from the input data in which geometrically closer nodes may not necessarily have a closer number. In addition, to perform the innermost loop IE in parallel, elements are colored in the entire computation region For this reason, a color will contain elements that are geometrically distant. Therefore, widely distributed elements in terms of geometry will appear throughout the innermost loop IE iterations. This geometric distance is also a cause of scattered memory access. Node number reordering and element blocking avoid those obstacles. And to decrease number of arrays referred in innermost loop, array merging is employed.

5.1.Node number reordering

In general, reordering is a method for changing the row or column order in a matrix to reduce bandwidth or avoid fill-in during matrix-vector multiplication [5][6][7]. However, the gradient computation kernel calculates element-by-element without matrices. Changing the order of a row or column is equivalent to changing a node number. Here node numbers are renumbered so that proximal nodes will have closer numbers.



(a) Node number reordering in geometry (b) By blocking, a block consists close set of elements

Fig.2. Schematic diagram of node number reordering and element blocking

Fig.2(a) illustrates this node renumbering concept. The left cube represents an entire calculated geometrical region that includes all the nodes. First, the region is divided into multiple subregions by segmenting the X, Y, and Z axes. This division is illustrated as small cubes. Although the axes have three components in Fig.2(a), each axis was actually segmented into 10 components by the pre-examination. Second, each subregion was numbered by the order of the X, Y, and Z axes. Next, each subregion was divided into an outer area and an inner area. Then, the nodes located in each subregion were renumbered from the first subregion so that the nodes included in the inner region would have smaller numbers and nodes included in the outer region would have larger numbers. By changing the node number as described above, closer sets of node in terms of geometry position are located near each other in memory.

5.2. Element blocking

In the original code, to perform the innermost loop IE in parallel, elements are colored in the entire computation region. Therefore, a color that contains distant elements with various positions will result in scattered memory access. Element blocking avoids such obstacles. Fig.2(b) shows a schematic diagram of the element blocking. The entire computation region is divided into blocks, and elements in a block are colored. In this method, each block is colored, i.e., a color consists of blocks that are not neighboring. This produces that many elements belong in a color and an innermost loop have sufficient iteration.

5.3. Array merging and final achieved result

In general, if numerous arrays are referred in a loop, cache hit rate and memory throughput will be worse. The gradient computation kernel references eight arrays for the innermost loop. Therefore, we tried merging arrays DNX, DNY, and DNZ and arrays FX, FY, and FZ. Table 1 shows the array merge patterns, performance

ratios, L1D cache miss ratios, and memory throughputs. Pattern A merged arrays DNX, DNY, and DNZ into array DNXYZ. Pattern B merged arrays FX, FY, and FZ into array FXYZ. Pattern C applied both merge patterns A and B. In merge pattern C, we obtained a L1D cache miss ratio of 3.37%, memory throughput of 38.8 Gbytes/s, and performance ratio of 4.41%, which is approximately 76% of the ideal performance 5.83%.

Table 1. Array merge patterns and measurement results

Merge pattern	Before merge	After Merge	Performance ratio	L1D cache miss ratio	Memory throughput [Gbyte/s]
A	DNX(9,N),DNY(9,N),DNZ(9,N)	DNXYZ(3,9,N)	3.95%	3.98%	35.7
B	FX(M), FY(M), FZ(M)	FXYZ(3,M)	4.05%	3.69%	35.9
C=A+B	DNX(9,N),DNY(9,N),DNZ(9,N) FX(M), FY(M), FZ(M)	DNXYZ(3,9,N) FXYZ(3,M)	4.41%	3.37%	38.8

6. Summary

We improved the gradient computation kernel of the finite element based fluid analysis code FrontFlow/blue on the K computer. The improvement methods included node number reordering and element blocking to localize memory access, a modified coloring method to avoid innermost loop overhead, and array merging to reduce the number of arrays referred in a loop. The new kernel performance is 2.7 times faster than that of the original code. These improvements will be useful not only for the gradient computation kernel of the FrontFlow/blue but also for other operations that have reference from element to node frequently appear for finite element method, finite volume method.

The results in this paper were obtained by early using period of the K computer at the RIKEN Advanced Institute for Computational Science.

Acknowledgements

We would like to thank Professor Chisachi Kato of the Institute of Industrial Science, The University of Tokyo, members of the Research and Development of Innovative Simulation Software project, The University of Tokyo, and colleagues in the Operations and Computer Technologies Div. RIKEN AICS.

References

[1] http://www.ciss-test.iis.u-tokyo.ac.jp/rss21/en/theme/multi/fluid/index.html

[2] Minami, K., Inoue, S., Tsutsumi, S., Maeda, T., Hasegawa, Y., Kuroda. A., Terai. M. and Yokokawa, M., “Performance Tuning and Evaluation of Sparse matrix-vector multiplication on the K computer”, Proc. of the High Performance Computing Symposium 2012, pp.23-31, 2012, (In Japanese).

[3] Maruyama, T., “SPARC64 VIIIfx: Fujitsu's New Generation Octo-core Processor for Peta Scale Computing”, Hot Chips Vol.21, 2009.

[4] http://www.cs.virginia.edu/stream/ref.html

[5] Tinney, W.F. and Walker, J.W., “Direct solutions of sparse network equations by optimally ordered triangular factorization”, Proc. IEEE Vol.55, pp.1801-1809, 1967.

[6] George, A., “Nested dissection of a regular finite-element mesh”, SIAM J. Numerical Analysis Vol.10, pp.345-363, 1973.

[7] Pinar, A. and Heath, T.M. Improving performance of sparse matrix-vector multiplication. Proc. ACM/IEEE Conference on Supercomputing, Portland, Oregon, 1999.